A new implementation of free-text search using a new parallel computer—the Connection Machine®—makes possible the application of exhaustive methods not previously feasible for large databases.

PARALLEL FREE-TEXT SEARCH ON THE CONNECTION MACHINE SYSTEM

CRAIG STANFILL and BREWSTER KAHLE

Locating information in natural-language documents contained in large databases is an important problem. Such databases may contain articles from wire services and newspapers, abstracts and full-text articles from journals and encyclopedias, as well as bibliographies and judicial rulings. Search facilities for such databases are provided by vendors like Dialog, Lexis, Nexis, Westlaw, Medlars, and VU-Text. In many communities, particularly legal and biomedical research, access to such databases has become critical. The size of such databases can be virtually unlimited: The Lexis database, for example, reportedly contains a total of 125 Gbytes of information.

The two general methods of accessing such databases are free-text search and keyword search: Free-text search systems allow the user to search the text of the documents for arbitrary combinations of words, whereas keyword systems require that indexers read all the documents in a database and assign them keywords from a controlled vocabulary. In the latter system, the user searches for information or documents using some combination of keywords from this controlled vocabulary. The relative merits of these systems continue to be debated [1, 11]. Advocates of manual indexing systems argue that free-text search systems suffer from poor recall (they fail to retrieve all relevant documents) and poor precision (they deliver too many irrelevant documents). Proponents of free-text search counter that techniques are available to alleviate these two problems and that manual indexing systems have problems of their own: specifically, that the manual indexing process is unreliable, based on the observation that different indexers will often assign radically different keywords to the same article.

Although techniques are available to improve the recall and precision of searches in free-text database systems, they are in some cases not widely used because they make computational demands beyond the practical limits of conventional computers.

The Connection Machine is a registered trademark of Thinking Machines Corporation.

© 1986 ACM 0001-0782/86/1200-1229 75¢

The usual method of organizing a free-text database is to construct an inverted index of all words it contains [4, 12]. To locate the documents containing a given word, the index for that word is located (via hashing, for example) and retrieved. Documents containing combinations of words are found by computing the intersection and union of indexes. However, inverting a database in this way often increases the storage utilization 200 percent over the raw text.

This article will take no stand on the issue of the relative merits of free-text versus controlled-vocabulary searching. Rather, we will present a new implementation of free-text searching that takes advantage of the possibilities offered by a massively parallel computer, the Connection Machine (CM), with up to 65,536 processing elements [3]. In this implementation, the representation of the documents in memory permits a very fast search for the presence of a word in a document. With the CM, each processing element stores between one and three documents; queries are then broadcast to the entire machine and the results collected. Because of the massive parallelism, the resulting system is fast enough to permit the application of exhaustive methods not previously feasible for large databases.

Two applications of this technology are discussed: The first is a benchmark of the CM as a query evaluator. In this application, a Boolean query is evaluated against a sample database. Using this measurement and taking into account the approximate performance of the I/O system, we estimate that the CM can evaluate a 20,000-term query against a 15 Gbyte database in 3 minutes, of which 2 minutes are devoted to I/O and 1 minute to computation. The second application, a prototype of an interactive document-retrieval system, uses a technique called relevance feedback to produce an interface that is fast and easy to use, and produces a high-quality search.

THE CONNECTION MACHINE SYSTEM

The Connection Machine System is a fine-grained, highly parallel computer whose current version consists of between one and four modules of 16,384 processing elements each, for a maximum configuration of 65,536 processing elements. Each of these elements has 4,096 bits of memory and a 1-bit-wide ALU (arithmetic and logical unit), so that adding two 32-bit numbers takes 32 machine cycles. Each module is controlled by a single microcontroller, which executes macroinstructions originating in a host computer. The modules can be operated in concert, running one program, or independently, running up to four different programs. A machine cycle (3 microcontroller states) takes about 750 ns, yielding a raw system throughput exceeding two billion 32-bit operations per second. When operating on data smaller than 32 bits, proportionally higher processing rates are obtained. For the document-retrieval system about to be described, the measured performance is approximately six billion operations per second.

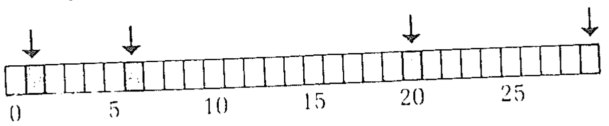
All processing elements execute the same program, with each operating on the contents of its own memory. The ALU has a context flag that allows processors to be selectively disabled. Communication is provided by three independent communication networks: a 1-bit-wide global-OR network, a 2-dimensional grid, and a 12-dimensional hypercube with 16 processors at each vertex. A global-OR network allows such operations as global-minimum, global-maximum, and global-OR to be performed in a time span proportional to the length of the operands: on the order of microseconds per bit. The grid allows instantaneous communication between nearest neighbors. The hypercube network supports full packet-switched communication between arbitrary processors. A disk system, which will allow the CM's full memory capacity of 32 Mbytes to be swapped in and out of mass storage in about 0.6 s, is currently under design. With the exception of the hypercube network and the disk system, the CM is similar to the Massively Parallel Processor (MPP) [9].

DATA STRUCTURE

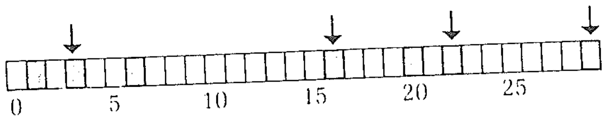
The starting point of the information-retrieval system is the representation of documents in the CM's memory. The basic requirement is to be able to very quickly test the machine's memory for the presence of a word. In addition, the representation must be reasonably compact. We selected a representation that was originally developed for spelling-correction dictionaries and is known as *surrogate coding* [2, 7, 8]. The advantage of surrogate coding is that probing a surrogate table for the presence of a word requires only AND-ing together a small number of bits; the disadvantages are that word order and proximity information are lost, and probes sometimes return false positives.

A surrogate table is a binary array k bits long. In the example that follows, we will use $k = 30$, although in actual implementations we have used values of 512 and 1024. The table is initialized to all zeros.

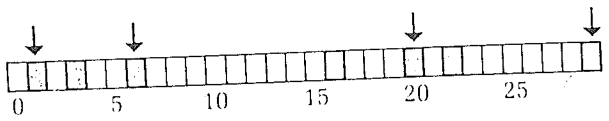
To insert a document into the table, i independent hash codes between 0 and $k - 1$ are generated. Here, we will use $i = 4$, although, in practice, i is usually between 10 and 30. If we insert the word "chemistry" into an empty table, and *hash*(*chemistry*) = (1, 6, 29, 20), setting these four bits to 1 produces the following table:



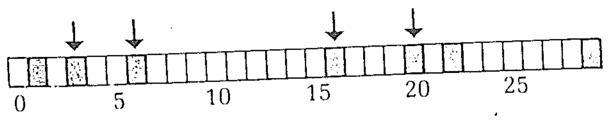
If we then insert a second word "biology" into the table, with *hash*(*biology*) = (16, 22, 29, 3), setting these four bits in the table produces the following layout. Note that one of these bits (29) was already set.



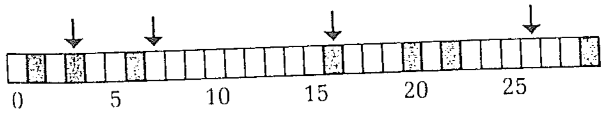
To probe for a word, we generate its *i* hash codes and then AND together the corresponding bits. To probe for "chemistry," we AND together bits 1, 6, 29, and 20. Since all four are 1, the probe returns PRESENT.



Suppose we probe for a second word, "physics," which was not inserted in the table. Assuming that *hash*(*physics*) = (3, 7, 16, 26), we check the table and find that bits 7 and 26 are 0. Thus, the probe returns ABSENT.



It is possible for a probe to return a false positive. In the case of *hash*(*mathematics*) = (3, 6, 16, 20), checking those four bits will show that all are 1. Thus, the probe returns PRESENT, even though "mathematics" was never inserted in the table.



The probability of a false positive depends on the size of the table (k), the number of words encoded in the table (w), and the number of bits that are set for each word (i). Decreasing w reduces the false-hit rate, but also increases storage needs. Increasing i, up to a point, also reduces the false-hit rate, but increases processing time. An analysis that will provide some guidance in making the most appropriate trade-off is presented next.

Analysis of the Data Structure

The probability of a false positive may be determined by finding the probability that a random bit is 1 as a function of k, w, and i, and then using this result to calculate the probability of i random bits all being 1. Given w words, and i bits set for each word, a total of $w \times i$ bits will be set. We will now calculate $P_1(w, i, k)$, the probability that a random bit in the table will be set. With a k-bit table T and a sequence S of $w \times i$ random integers uniformly distributed between 0 and $k - 1$, an element T_j of T is 1 if and only if j occurs in S; otherwise, it is 0. Therefore,

$$P_1(w, i, k) = 1 - P_0(w, i, k)$$

where $P_0(w, i, k)$ is the probability that a random number j, uniformly distributed between 1 and $k - 1$, does not occur in the sequence S. This gives us the following two formulations:

$$P_0(w, i, k) = \left(\frac{k-1}{k}\right)^{wi}$$

$$P_1(w, i, k) = 1 - \left(\frac{k-1}{k}\right)^{wi}$$

We now need to determine the probability that i random bits are all 1: We will call this $P_{false}(w, i, k)$. This is the probability of a false hit: That is, the probability that a word not present in the document used to create the table will return PRESENT when it is probed for.[1]

$$P_{false}(w, i, k) = P_1(w, i, k)^i = \left(1 - \left(\frac{k-1}{k}\right)^{wi}\right)^i$$

Table I shows some values of P_{false}, assuming $k = 512$ and $i = 10$. The important fact to remember about Table I (p. 1232) is that a small change in w causes a large change in P_{false}: That is, the false-hit rate may be dramatically reduced by a small decrease in the number of words in each table. Depending on the false-hit rate that is considered acceptable in any particular application, the values of w, i, and k can be tuned to yield optimal system performance.

APPLICATION

In practice, the application of this algorithm is slightly more complex than suggested above. First, each processor in the CM contains 4096 bits of memory, some of which are used as scratch memory. This allows six tables of 512 bits, or three tables of

[1] This calculation makes the simplifying assumption that the result of probing different locations in the table is statistically independent. This is not strictly correct, but for useful values of w, i, and k, it is not a bad assumption. Tom Kraay [5] has derived an exact form of P_{false} and shown it to be very close to our form for nonsmall values of w and k.

1024 bits. Moreover, as the above discussion indicates, it is necessary to limit the number of words in each table. For a 512-bit table, a limit of 15 to 30 words is reasonable. If a document contains more than the allotted number of words, several tables must be used for each document. Thus, if 30 words are being put into each table and a 90-word document is encountered, it is necessary to use three tables, which are generally put in three different processors. Such a set of tables is called a *chain*, where each chain reports to a designated *head processor*.

Probes for words are never used in isolation, but rather as components of Boolean queries (where they are combined via Boolean connectives), or as components of simple queries (where each word is given a numerical weight).

Boolean Queries

Boolean queries are built from primitive word tests that are combined via the Boolean connectives AND, OR, and NOT. To compute the OR of a set of word tests, each table is probed independently and the results for each table OR-ed together. These results are then passed on to the head processor of the chain and OR-ed together. If we had two documents, the first containing the words "A few words that might be in some document" and the second containing the words "Some other words that a document might well contain," and each document were broken up into three tables, considering the query

might OR right

would involve first probing each table for both words. Then, the process of OR-ing together the results of the probes would produce the following results:

A few words	*that might be*	*in some document*
absent	present	absent

Some other words	*that a document*	*might well contain*
absent	absent	absent

Passing the results to the head of each chain and OR-ing them together yield a hit on the first document.

A few words	*that might be*	*in some document*
present		

Some other words	*that a document*	*might well contain*
absent		

TABLE I. Probability of a False Hit

w = Words per table
i = Bits set for each word, fixed at 10
k = Bits per table, fixed at 512
P_{false} = Probability of a false positive being returned when probing one table for one word

w	P_{false}
5	4.91×10^{-11}
10	3.12×10^{-8}
15	1.13×10^{-6}
20	1.26×10^{-5}
25	7.46×10^{-5}
30	2.96×10^{-4}

To compute the AND of a set of primitive word tests, each table is probed independently, and the number of probes that return PRESENT is recorded for each table. The results are then passed to the head processor of the chain and summed. If the total is equal to the total number of terms in the AND clause, the clause returns true. This method requires that each word in the document occur in only one table. In the query

some AND words AND in AND a AND document

we first probe for each word, counting the number present in each table:

A few words	*that might be*	*in some document*
2	0	3

Some other words	*that a document*	*might well contain*
2	2	0

Sending the results to the head processor and adding reveal that the first document had 5/5 of the words and therefore satisfies the query, whereas the second document had only 4/5 and therefore fails the test.

A few words	*that might be*	*in some document*
5		

Some other words	*that a document*	*might well contain*
4		

The set of documents matching a particular query is extracted from the database using a global-maximum operation. Each document has a document number that is stored in its head processor. The query is executed, and processing elements that do not record a hit are masked out. The global-maximum operation is then used to find the largest numbered document in an active processor. That processor is then masked out, and the process continues until all selected documents have been extracted.

Simple Queries

In simple queries, point values are assigned to a set of words, and each document in the database is then scored by totaling the scores of the words it contains. This is done by adding up the score for the words in each processor and then totaling the scores of the processors in each chain. With the following simple query

some	2
words	2
in	1
a	1
document	4

we first probe each table for each of the five words, summing the scores of the words that are found:

A few words	*that might be*	*in some document*
3	0	7

Some other words	*that a document*	*might well contain*
4	5	0

We then add the scores for each table in a chain, yielding a score of 10 for the first document and 9 for the second.

A few words	*that might be*	*in some document*
10		

Some other words	*that a document*	*might well contain*
9		

The documents in the database that are assigned nonzero scores are retrieved best-to-worst. The global-maximum operation is used to find the processor in the CM with the highest total score; that processor is masked out, and the process repeated to extract the second highest score, and so forth. The process can be halted when a specified number of documents has been retrieved, or when only documents with a score below a specified threshold remain in the machine.

A BENCHMARK

In September 1985, a benchmark of the Boolean query algorithm was performed on a 16,384-processor prototype of the CM. The test database consisted of 31,994 documents, totaling 18 Mbytes. A document consisted of several fixed-format numerical fields, totaling 100 bits, and two variable-length text fields. Each virtual processor in the CM had two tables of 512 bits each, and the packing density was limited to 35 words per table. Several trial queries, containing between 25 and 20,000 atomic terms, were evaluated. The times required to execute the queries varied between 0.004 s for a 25-term query and 0.295 s for a 20,000-term query. For a full-sized CM with 65,536 processors, execution times would remain the same, while the database could grow to 128,000 documents totaling 71 Mbytes.

Databases larger than 71 Mbytes will require the use of the swapping disk to page additional segments of the database into the CM's memory. Swapping in the data should require approximately 0.6 s per database segment. From then on, evaluating a query again takes between 0.004 s (25 terms) and 0.295 s (20,000 terms) per database segment. Using these figures, the time required to search a 15-Gbyte database should vary between 2 min. for a 25-term query and 3 min. for a 20,000-term query—a substantial improvement over existing search systems. The results of this benchmark are summarized in Table II (p. 1234).

SEARCH STRATEGIES

To date, the methods used to search large databases have been limited by the nature of the sequential computers available to perform them. Since the basic problem is that an exhaustive search may take a very long time, on-line searches on sequential machines have traditionally required nonexhaustive methods such as searching inverted indexes. Although inverted indexes suffice for short queries, as the number of terms in the queries grows they become impractical. In particular, as the number of search terms becomes large, the number of documents having at least one of the terms approaches the size of the database itself. The CM's ability to conduct an exhaustive parallel search in a small amount of time changes the situation radically. Search strategies that have been discussed for some time, but not commercialized, have suddenly become practical.

In evaluating search strategy performance, we use three standard criteria. The first is a pair of quantitative parameters that together measure the quality of the search: *precision*, which is the proportion of retrieved documents that are relevant; and *recall*, which is the proportion of relevant documents in the entire database that are retrieved. The second criterion, *ease of use*, involves a determination of how quickly a user can learn to use the system, and how

TABLE II. Results of the Boolean Query Benchmark

Measured benchmark and projected times

	Query size				
	25 terms	100 terms	1,000 terms	10,000 terms	20,000 terms
Load, one batch (s)	0.567	0.567	0.567	0.567	0.567
Execute query, one batch (s)	0.004	0.005	0.022	0.149	0.295
Total time per batch (s)	0.571	0.572	0.589	0.716	0.862
Hits per batch	22	22	25	50	68
Total time for 15 Gbytes (min.)	2.0	2.1	2.1	2.5	3.0

	Benchmark batch size (16K processors)	Full batch size (64K processors)	Full database size (211 batches)
Raw data	18 Mbytes	71 Mbytes	15 Gbytes
Compressed data	8 Mbytes	32 Mbytes	7 Gbytes
Documents	31,994	128,000	27,000,000

easy the system is to use once it has been mastered. Finally, there is *response time*: If a search system is too slow, it cannot be used for interactive search.

Boolean Queries

Although Boolean queries are typically composed of word tests plus the connectives AND, OR, and NOT, many variations on Boolean queries can be implemented by adding additional operators. Boolean queries implemented with inverted indexes are currently the primary method for searching large full-text databases. They are limited in the quality of the search they provide, and users find them clumsy, but they do deliver adequate performance.

However, there is a trade-off between recall and precision that limits the quality of Boolean searches on large databases [1]. Searching a database for documents containing a single word may lead to low recall because there is no guarantee that all relevant documents will use that word. In addition, it is likely that a large number of irrelevant documents will be retrieved, leading to low precision. Searching for several words together aggravates one or the other of these problems: If the searcher looks for any of several words (a disjunctive query), recall improves but precision goes down; on the other hand, if the searcher looks for documents containing all of several words (a conjunctive query), precision improves but recall suffers. For a large database, this means that the searcher may have to choose between missing important information and reading thousands of irrelevant documents.

There are other problems with the use of Boolean queries for full-text search. First, the user is playing a guessing game when trying to anticipate which words the authors of the documents he or she is interested in might have used. Second, even if the user guesses the words, he or she has to figure out which connectives to use to avoid getting either too much or too little data; this often involves several iterations as the query is debugged. Moreover, the syntax of Boolean queries is complex, making the systems difficult to learn.

As far as performance is concerned, Boolean search systems employing inverted indexes are fast enough to be commercially successful. Nonetheless, the problems with search quality and usability are severe, and there is wide agreement that something better is needed [1].

Simple Queries

An alternative to a Boolean search strategy is to use simple queries [12]. A simple query consists of a set of words, each of which is assigned a point value. Every document in the database is scored (for potential retrieval value) by adding up the point values for the words it contains. The total score can then be normalized to take into account the size of the document and the length of the query.

Simple queries are easier to use than Boolean queries because the user does not need to decide which connectives to use, as there are none, and it is not necessary to learn a complex query language, as the query consists of a list of words. However, the user does still need to guess which words to use in the query.

Relevance Feedback

Relevance feedback ([10, 12]) constructs simple queries from the texts of documents that have already been judged relevant. After some search method has been used to locate a small set of possibly relevant documents, the user scans these documents and marks any documents that are obviously relevant as *good*, and any that are obviously irrelevant as *bad*. The text of the marked documents is then scanned for words, and a simple query is constructed from these words. The more good documents a word occurs in, the higher its score.

Since simple queries built using this technique may contain hundreds of terms, its practical application on large databases will require a very powerful machine. Applying the CM to this task, since it has both sufficient processing power and sufficient I/O bandwidth, results in an excellent search in terms of quality, ease of use, and performance, as discussed in detail below.

THE SEED DOCUMENT SEARCH SYSTEM

The combined use of simple queries and the relevance feedback technique was tested in a prototype system running on the CM. The test database consists of approximately 16,000 articles from the Reuters wire service totaling approximately 32 Mbytes and dating from December 17, 1985, to February 1, 1986. In this test system, the text is run through the Context® Text Indexer [6], which does simple linguistic and semantic analysis to determine which words may be dropped from the document without affecting its retrievability. All words not dropped by the Indexer are encoded in the surrogate tables (described on p. 1232) and stored in a 16,384 processing element version of the CM that has 4,096 bits of storage per element. The interface is mouse driven and runs on a Symbolics 3600-series® Lisp machine. The full text of the data is stored on the Lisp machine's disk system.

To begin the search, the user types in a set of *seed words*. The search system assigns these words scores as a function of their frequency in the database as a whole, and evaluates the resulting simple query. Evaluating this simple query and retrieving the results takes approximately 20 ms. The results of this initial query are displayed as a list of citations[2] that the user can browse through by clicking a mouse. For example, to look for reports of former Philippine President Ferdinand Marcos's hidden assets in the United States, one might enter the seed words "`marcos hidden wealth`," which would produce the set of articles shown in Figure 1 (p. 1236).

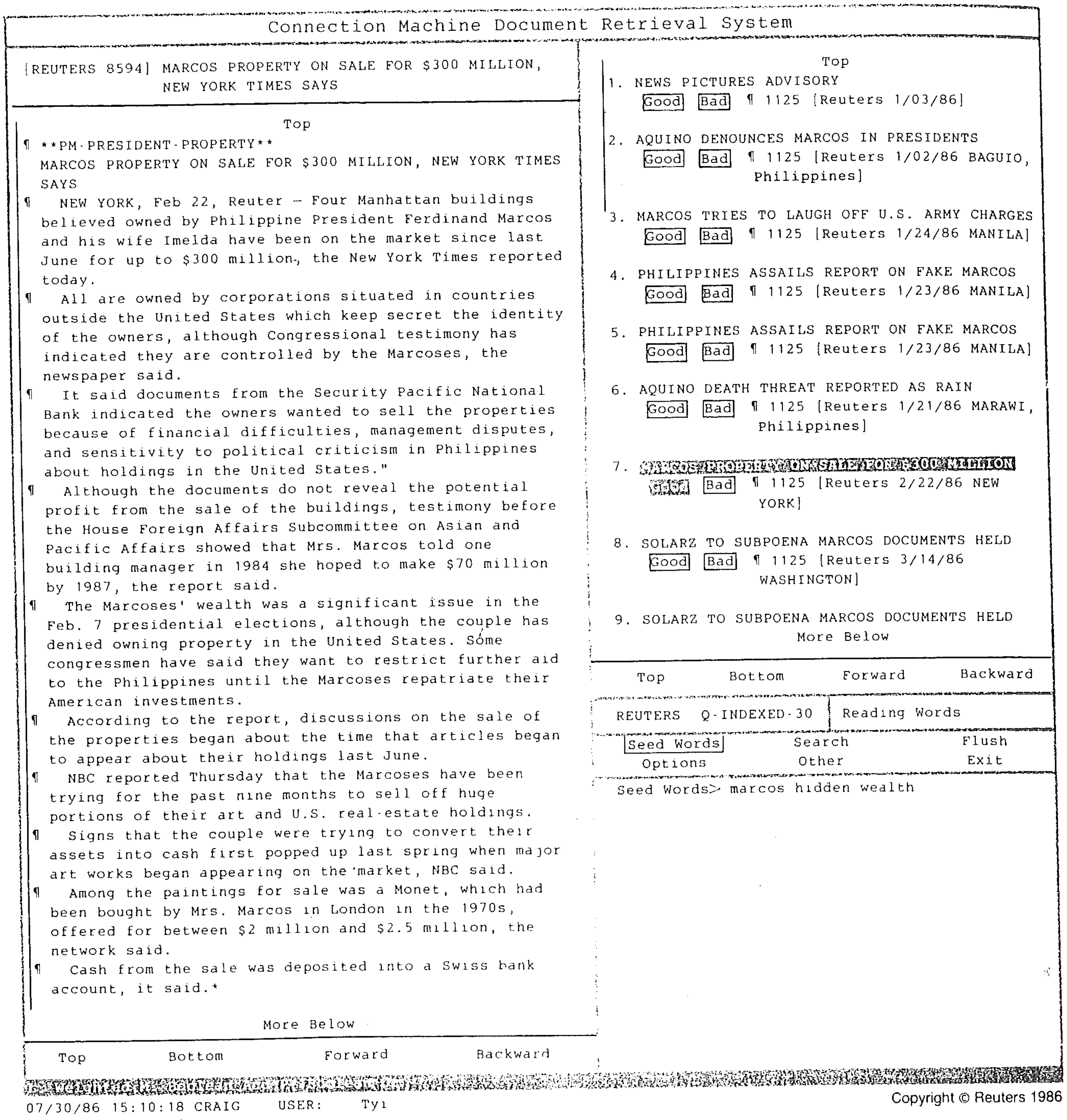
[2] For the Reuters news database, the citations include headlines plus dates and places of filing.

Context is a trademark of Thinking Machines Corporation.
Symbolics 3600 is a trademark of Symbolics, Inc.

During the process of perusing the initial set of retrieved documents, the user begins a second phase of search. When a document from this set is determined to be relevant to the topic, the user clicks a mouse on a box labeled `Good`. The search system notes the words contained in that document and assigns them scores for use in the next round of search. The user may mark as many documents as desired. When the search system is invoked again, all the words in the documents that the user has marked `Good` go into the simple query. The user may also mark documents `Bad`, which causes words that are in *bad* documents but not in *good* ones to be assigned inhibitory weight. This method yields a query composed of several hundred words. The results of such a search are typically available in approximately 1 s, which is largely consumed by overhead in the interface and support system; a search that generates 200 terms and retrieves 20 candidate documents uses only 60 ms of CM time. The results of this second phase (or *seed document*) search are usually much better in terms of higher recall and better precision than those attained in the first phase. In our example (Figure 1), we mark the document entitled "`MARCOS PROPERTY ON SALE FOR $300 MILLION, NEW YORK TIMES SAYS`" as a seed document (i.e., `Good`) and perform a second search that yields the set of documents shown in Figure 2 (p. 1237). The user may repeat the process as often as desired, with search results becoming better and better as the query is refined.

Relevance feedback leads to both high precision and high recall as a result of the large number of words employed in the search process. One word, taken by itself, conveys little information, but 200, taken as a whole, convey a great deal. Only highly relevant documents will use a high proportion of this set of 200 terms.

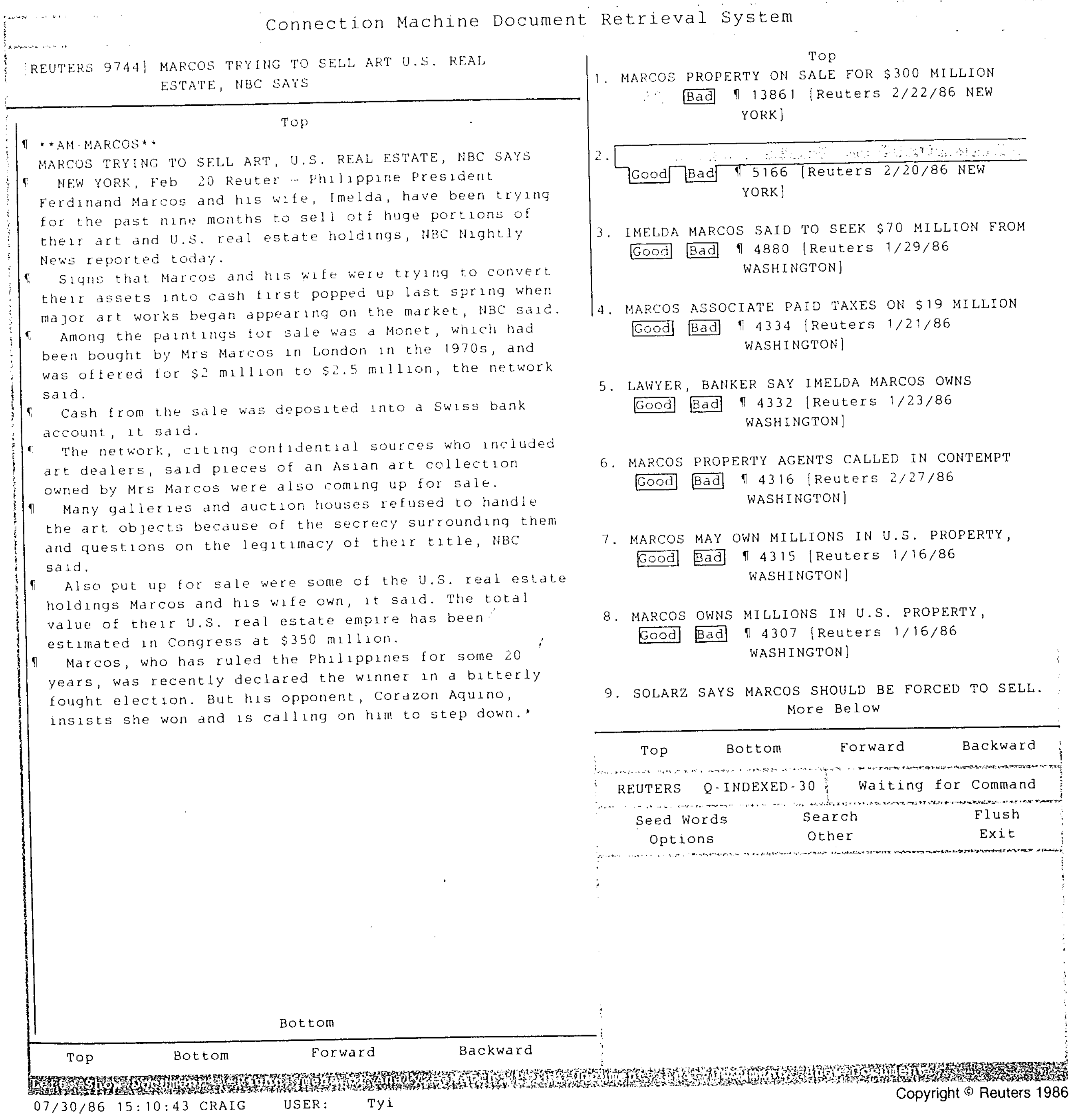
To test the effectiveness of relevance feedback vis-à-vis the traditional Boolean search, we experimented with two separate retrieval tasks: locating reports of a summit meeting between Jordan and Syria in late December 1985, and retrieving reports that Ferdinand Marcos had hidden some of his assets in the United States (our sample search shown in Figures 1 and 2). First, a very broad Boolean query was executed to search the database: In the first case, this query consisted of the word "Syria;" in the second, it was "Marcos." These two queries retrieved 288 and 218 documents, respectively. These sets of candidate documents were then browsed through to find relevant documents (14 and



```
                    Connection Machine Document Retrieval System

[REUTERS 8594] MARCOS PROPERTY ON SALE FOR $300 MILLION,        Top
               NEW YORK TIMES SAYS                           1. NEWS PICTURES ADVISORY
                                                                [Good] [Bad] ¶ 1125 [Reuters 1/03/86]
                         Top
¶ **PM-PRESIDENT-PROPERTY**                                  2. AQUINO DENOUNCES MARCOS IN PRESIDENTS
  MARCOS PROPERTY ON SALE FOR $300 MILLION, NEW YORK TIMES      [Good] [Bad] ¶ 1125 [Reuters 1/02/86 BAGUIO,
  SAYS                                                                Philippines]
¶   NEW YORK, Feb 22, Reuter - Four Manhattan buildings
  believed owned by Philippine President Ferdinand Marcos    3. MARCOS TRIES TO LAUGH OFF U.S. ARMY CHARGES
  and his wife Imelda have been on the market since last        [Good] [Bad] ¶ 1125 [Reuters 1/24/86 MANILA]
  June for up to $300 million-, the New York Times reported
  today.                                                     4. PHILIPPINES ASSAILS REPORT ON FAKE MARCOS
¶   All are owned by corporations situated in countries         [Good] [Bad] ¶ 1125 [Reuters 1/23/86 MANILA]
  outside the United States which keep secret the identity
  of the owners, although Congressional testimony has        5. PHILIPPINES ASSAILS REPORT ON FAKE MARCOS
  indicated they are controlled by the Marcoses, the            [Good] [Bad] ¶ 1125 [Reuters 1/23/86 MANILA]
  newspaper said.
¶   It said documents from the Security Pacific National     6. AQUINO DEATH THREAT REPORTED AS RAIN
  Bank indicated the owners wanted to sell the properties       [Good] [Bad] ¶ 1125 [Reuters 1/21/86 MARAWI,
  because of financial difficulties, management disputes,             Philippines]
  and sensitivity to political criticism in Philippines
  about holdings in the United States."                      7. MARCOS PROPERTY ON SALE FOR $300 MILLION
¶   Although the documents do not reveal the potential          [Good] [Bad] ¶ 1125 [Reuters 2/22/86 NEW
  profit from the sale of the buildings, testimony before             YORK]
  the House Foreign Affairs Subcommittee on Asian and
  Pacific Affairs showed that Mrs. Marcos told one           8. SOLARZ TO SUBPOENA MARCOS DOCUMENTS HELD
  building manager in 1984 she hoped to make $70 million        [Good] [Bad] ¶ 1125 [Reuters 3/14/86
  by 1987, the report said.                                           WASHINGTON]
¶   The Marcoses' wealth was a significant issue in the
  Feb. 7 presidential elections, although the couple has     9. SOLARZ TO SUBPOENA MARCOS DOCUMENTS HELD
  denied owning property in the United States. Some                       More Below
  congressmen have said they want to restrict further aid
  to the Philippines until the Marcoses repatriate their       Top       Bottom       Forward       Backward
  American investments.
¶   According to the report, discussions on the sale of       REUTERS  Q-INDEXED-30  | Reading Words
  the properties began about the time that articles began
  to appear about their holdings last June.                   [Seed Words]       Search          Flush
¶   NBC reported Thursday that the Marcoses have been           Options          Other           Exit
  trying for the past nine months to sell off huge
  portions of their art and U.S. real-estate holdings.        Seed Words> marcos hidden wealth
¶   Signs that the couple were trying to convert their
  assets into cash first popped up last spring when major
  art works began appearing on the market, NBC said.
¶   Among the paintings for sale was a Monet, which had
  been bought by Mrs. Marcos in London in the 1970s,
  offered for between $2 million and $2.5 million, the
  network said.
¶   Cash from the sale was deposited into a Swiss bank
  account, it said.*

                    More Below

  Top        Bottom        Forward        Backward
[illegible]
07/30/86 15:10:18 CRAIG     USER:     Tyi                        Copyright © Reuters 1986
```

FIGURE 1. Seed Word Phase of a Search

12), which were presumed to be the entire set of relevant documents in the database. Next, for comparative purposes, *conjunctive Boolean queries* of two terms were executed: For the first task, the query was "Jordan and Syria;" for the second, it was "Marcos and Wealth." The results of these queries were tabulated according to recall and precision values. Third, a *simple query* was executed for the two respective tests—"Jordan Syria summit" and "Marcos Hidden Wealth"—and recall and precision rates were tabulated for different subsets of the query result. In the first test, three tabulations were made for the best 10, 20, and 30 documents retrieved; in the second, only one tabulation was made



```
Connection Machine Document Retrieval System

[REUTERS 9744] MARCOS TRYING TO SELL ART U.S. REAL
               ESTATE, NBC SAYS

                         Top
¶ **AM-MARCOS**
MARCOS TRYING TO SELL ART, U.S. REAL ESTATE, NBC SAYS
¶   NEW YORK, Feb  20 Reuter - Philippine President
Ferdinand Marcos and his wife, Imelda, have been trying
for the past nine months to sell off huge portions of
their art and U.S. real estate holdings, NBC Nightly
News reported today.
¶   Signs that Marcos and his wife were trying to convert
their assets into cash first popped up last spring when
major art works began appearing on the market, NBC said.
¶   Among the paintings for sale was a Monet, which had
been bought by Mrs Marcos in London in the 1970s, and
was offered for $2 million to $2.5 million, the network
said.
¶   Cash from the sale was deposited into a Swiss bank
account, it said.
¶   The network, citing confidential sources who included
art dealers, said pieces of an Asian art collection
owned by Mrs Marcos were also coming up for sale.
¶   Many galleries and auction houses refused to handle
the art objects because of the secrecy surrounding them
and questions on the legitimacy of their title, NBC
said.
¶   Also put up for sale were some of the U.S. real estate
holdings Marcos and his wife own, it said. The total
value of their U.S. real estate empire has been
estimated in Congress at $350 million.
¶   Marcos, who has ruled the Philippines for some 20
years, was recently declared the winner in a bitterly
fought election. But his opponent, Corazon Aquino,
insists she won and is calling on him to step down.*

                        Bottom
 Top        Bottom        Forward        Backward

07/30/86 15:10:43 CRAIG     USER:     Tyi

                          Top
1. MARCOS PROPERTY ON SALE FOR $300 MILLION
        [Bad]  ¶ 13861 [Reuters 2/22/86 NEW
                 YORK]

2. [illegible]
   [Good] [Bad]  ¶ 5166 [Reuters 2/20/86 NEW
                 YORK]

3. IMELDA MARCOS SAID TO SEEK $70 MILLION FROM
   [Good] [Bad]  ¶ 4880 [Reuters 1/29/86
                 WASHINGTON]

4. MARCOS ASSOCIATE PAID TAXES ON $19 MILLION
   [Good] [Bad]  ¶ 4334 [Reuters 1/21/86
                 WASHINGTON]

5. LAWYER, BANKER SAY IMELDA MARCOS OWNS
   [Good] [Bad]  ¶ 4332 [Reuters 1/23/86
                 WASHINGTON]

6. MARCOS PROPERTY AGENTS CALLED IN CONTEMPT
   [Good] [Bad]  ¶ 4316 [Reuters 2/27/86
                 WASHINGTON]

7. MARCOS MAY OWN MILLIONS IN U.S. PROPERTY,
   [Good] [Bad]  ¶ 4315 [Reuters 1/16/86
                 WASHINGTON]

8. MARCOS OWNS MILLIONS IN U.S. PROPERTY,
   [Good] [Bad]  ¶ 4307 [Reuters 1/16/86
                 WASHINGTON]

9. SOLARZ SAYS MARCOS SHOULD BE FORCED TO SELL.
                      More Below

 Top        Bottom        Forward        Backward
 REUTERS   Q-INDEXED-30 |   Waiting for Command
  Seed Words         Search          Flush
   Options           Other           Exit

                              Copyright © Reuters 1986
```

FIGURE 2. Seed Document Phase of a Search

as only 9 documents in total were retrieved. The fourth phase of the experiment applied *seed document search* to the set of documents produced by the simple query used in phase three. A small number of the retrieved documents were marked Good (in experiment 1 (Syria), a single document; in experiment 2 (Marcos), three documents), and a search was triggered. Again, the results were tabulated for recall and precision, taking several subsets of the query result. In the fifth phase of experiment 1, each of the top 10 documents retrieved in the previous phase (i.e., by the first seed document search) was marked either Good or Bad; the test was repeated and the results tabulated.

The results of testing the techniques against the two test cases are shown in Figures 3 and 4. The figures show that the simple queries were comparable in recall and precision to Boolean queries, while the large queries generated by relevance feedback in the seed document phase of the search were significantly better in both recall and precision, than either the simple query alone or the Boolean query. This confirms previous results on the effectiveness of relevance feedback for purposes of document retrieval [10] and the problems encountered with Boolean searches [1].

The interface to our document-retrieval system is extremely simple to use. There is no query language to learn: The user need only know how to enter a list of words, browse through articles, mark them as Good or Bad, and initiate a new search—a 5-minute training session at most. In addition, the search does not involve an extensive guessing game. The initial phase, which involves locating a small initial set of relevant documents, does require a small amount

	Query	Recall	Precision
A	Disjunction, 3 terms	1.00	0.05
B	Conjunction, 2 terms	0.71	0.27
C1	Simple, 3 terms, first 10	0.36	0.50
C2	Simple, 3 terms, first 20	0.57	0.40
C3	Simple, 3 terms, first 30	0.71	0.33
D1	Mark 1, first 10	0.57	0.80
D2	Mark 1, first 20	0.86	0.60
D3	Mark 1, first 30	0.93	0.43
E1	Mark 10, first 10	0.71	1.00
D2	Mark 10, first 20	0.86	0.60
E3	Mark 10, first 30	1.00	0.47

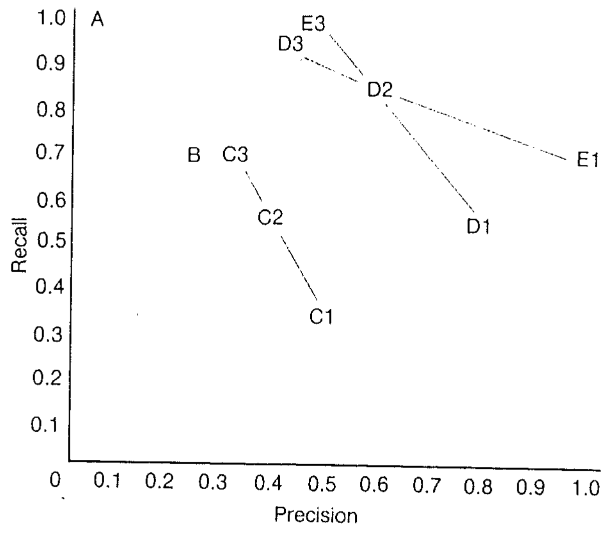


FIGURE 3. First Run of Retrieval Experiment

	Query	Recall	Precision
A	Single term	1.00	0.06
B	Conjunction, 2 terms	0.25	0.30
C	Conjunction, 2 other terms	0.17	0.33
D	Simple, 2 terms	0.25	0.30
E1	Mark 3, first 10	0.67	0.80
E2	Mark 3, first 20	0.92	0.55
E3	Mark 3, first 30	1.00	0.40

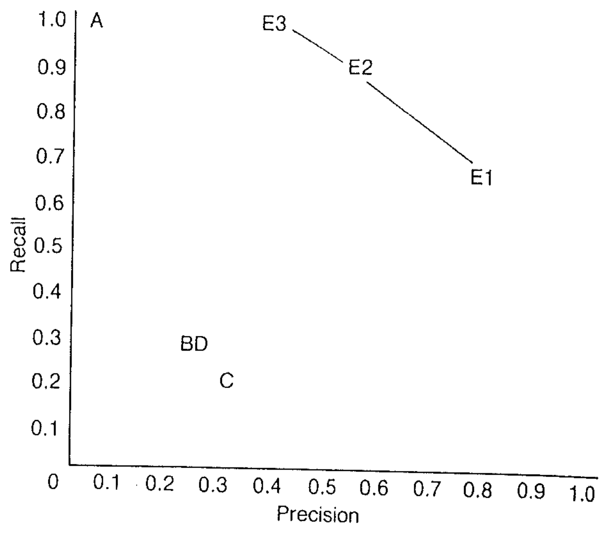


FIGURE 4. Second Run of Retrieval Experiment

of guesswork, but since the aim here is not high recall, the user is not required to be comprehensive (i.e., to think of every possible word an author might have used). Subsequent iterations do not require this sort of guessing, either, and reactions from people who have used the system have been extremely good.

Finally, the search system itself is extremely fast: For the sample database we have been working with, the response time is about a second. In general, only 60 ms of CM time are required to find the 20 best hits for a 200-term query, which indicates that the basic throughput of the machine as a search engine should be adequate to support a significant number of users on a large database.

CONCLUSION

Using the CM, it is possible to perform a fast parallel search of a database. For a database of 18 Mbytes (31,994 documents) on a 16,384 element machine, the measured time to execute a query varies from 0.004 s for a Boolean query with 25 terms to 0.295 s for a Boolean query with 20,000 terms (compute time only). For a 15-Gbyte database, the estimated time to execute a query varies between 2 min. for a Boolean query with 25 terms to 3 min. for a Boolean query of 20,000 terms (compute plus I/O time). It is also possible to search a database using simple queries and achieve similarly high search speeds: 60 ms for a 200-term query on a 112 Mbyte database.

Using these facilities, it is possible to implement database search techniques that are not feasible given sequential machines and inverted databases. A demonstration system utilizing relevance feedback has been written and tested. It combines high precision and recall with ease of use and fast response, and represents an advance over existing free-text database search technology.

Acknowledgments. Many thanks to Jim Bailey and Dave Waltz for guidance in this project, to Steven Smith for work on the user interface, and to the members of the Connection Machine System Group for providing the hardware. Special thanks to Tom Kraay for suggesting the project and to Reuters for making their wire service available to us.

REFERENCES

1. Blair, D.C., and Maron, M.E. An evaluation of retrieval effectiveness for a full-text document-retrieval system. *Commun. ACM 28*, 3 (Mar. 1985), 289–299.
2. Dodds, D.J. Reducing dictionary size by using a hashing technique. *Commun. ACM 25*, 6 (June 1982), 368–370.
3. Hillis, D. *The Connection Machine*. MIT Press, Cambridge, Mass., 1985.
4. James, G. *Document Databases*. Van Nostrand Reinhold, New York, 1985.
5. Kraay, T. Unpublished manuscript. MRJ, Fairfax, Va., Apr. 1986.
6. Mott, P.M., Waltz, D.L., Resnikoff, H.L., and Robertson, G.G. Automatic indexing of text. Tech. Rep. 86-1, Thinking Machines Corp., Cambridge, Mass., Jan. 1986.
7. Nix, R. Experience with a space efficient way to store a dictionary. *Commun. ACM 24*, 5 (May 1981), 297–298.
8. Peterson, J.L. Computer programs for detecting and correcting spelling errors. *Commun. ACM 23*, 12 (Dec. 1980), 676–687.
9. Potter, J.L. *The Massively Parallel Processor*. J.L. Potter, Ed. MIT Press, Cambridge, Mass., 1985.
10. Salton, G. *The SMART Retrieval System—Experiment in Automatic Document Processing*. Prentice-Hall, Englewood Cliffs, N.J., 1971.
11. Salton, G. Another look at automatic text-retrieval systems. *Commun. ACM 29*, 7 (July 1986), 648–656.
12. van Rijsbergen, C.J. *Information Retrieval*. 2nd ed. Butterworths, London, 1979.

CR Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures—*single-instruction-stream, multiple-data-stream processors (SIMD)*; H.3.2 [**Information Storage and Retrieval**]: Information Storage; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval
General Terms: Design, Performance
Additional Key Words and Phrases: Connection Machine, free-text databases, parallel computing, relevance feedback

Received 4/86; accepted 6/86; revised 8/86

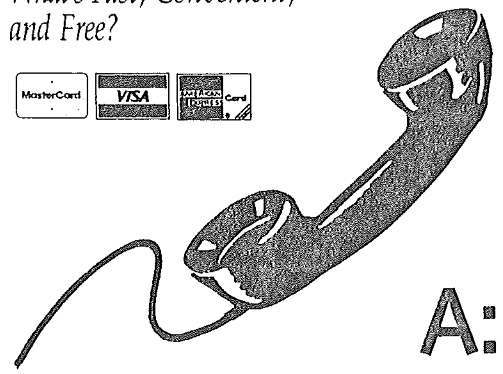
Authors' Present Address: Craig Stanfill and Brewster Kahle, Thinking Machines Corporation, 245 First Street, Cambridge, MA 02142.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

Q:

What's Fast, Convenient, and Free?

MasterCard VISA American Express Card



A:

ACM's "Order Express" Service for ACM Publications.

1-800-342-6626

Your credit card and our toll free number provide quick fulfillment of your orders.

- Journals
- Conference Proceedings
- SIG Newsletters
- SIGGRAPH VIDEO REVIEW
- "Computers in your Life" (An Introductory Film from ACM)

For Inquiries and other Customer Service call: *(301) 528-4144*.

acm ASSOCIATION FOR COMPUTING MACHINERY